Homework 3: Fuzzing and Delta Debugging

Troy Diaz

1

Explain the cause of the bug in the source code of the sort function (that is, the reason why mysort fails on min_failing_test.txt).

Inside min_failing_test.txt, it contains non-numeric characters instead of integers, “紊翘酋嗜是跷衔翁跷仁问跆蜗饰踟赏瓮跛锨甚腿似跷腿瓮仙硝掏屯术稳仕3”. The program, mysort.c, expects an input file with an integer per line, but the input contains “garbage” or bad characters. This suggests that the program mysort.c doesn’t validate the input before sorting.

In the code:

mysort.c

```
  while (1) {
      if (fgets(s, MAXLEN, fp) == NULL)
          break;
      x = (Link*) malloc(sizeof(Link));
      if (x == NULL) {
          printf("Error: malloc failed\n");
          exit(1);
      }
      x->data = atoi(s); // Here this converts the input string to an integer
      x->next = head;
      head = x;
  }
```

The function atoi() does not return errors if the input is invalid. For example, if s were to be 紊翘酋嗜是跷衔翁跷仁问跆蜗饰踟赏瓮跛锨甚腿似跷腿瓮仙硝掏屯术稳仕3, then atoi() returns 0. This means that invalid data gets inserted into the linked list without being validated.

terminal

```
 ➜  hw3 git:(main) ✗ ./mysort min_failing_test.txt 3
0
```

Also, there may be an infinite loop that is in mysort.c. If *y->data* or *z->data* contains garbage values, the logic of this snippet breaks. This could be caused by incorrect swaps, invalid values inside the linked list, and misaligned pointer updates which can cause the program to reference memory incorrectly. If the sorting algorithm assumes all data is valid, then it will fail to reach termination conditions.

mysort.c

```
            if (y->data >= z->data) {
              t = z->next;
              changed = TRUE;
              y->next = t;
              z->next = y;
              if (p == NULL)
                  x = z;
              else
                  p->next = z;
              p = z;
              z = t;
            } else {
              p = y;
              y = z;
              z = y->next;
            }
```

2

Characterize all input lists on which the sort function will fail because of the reason reported in (1) above.

The function atoi() returns 0 in a failure case and since mysort.c assumes input values are valid integers, then the following inputs will fail.

The string 紊翘酋嗜是跷衔翁跷仁问跆蜗饰跏赏瓮跛锨甚腿似跷腿瓮仙硝掏屯术稳仕 is not a valid integer and atoi() will return 0.

input.txt

```
2
98
紊翘酋嗜是跷衔翁跷仁问跆蜗饰跏赏瓮跛锨甚腿似跷腿瓮仙硝掏屯术稳仕
29
```

Leading or trailing non-numeric characters will also fail under atoi(), which may return 123 or garbage values under these conditions.

input.txt
123abc 456 789

Duplicate entries may cause an infinite loop as discussed earlier. The next pointers could form a cycle.

input.txt
20 40 30 40

Sorted lists could also cause issues, since the condition to sort is >=. The output may be an incorrect swap.

input.txt
1 4 4 6

If numbers are out of range, then atoi() could return overflowed values that lead to incorrect swaps.

input.txt
314159265359 -314159265359 20

3

Report a fix for the bug.

To fix the issues in mysort.c values must be validated before inserting into the linked list. Validated inputs should be integers. Instead of using atoi() we'll use **strtol().** This will stop

non-numeric characters from being inserted, ensure only valid integers are being sorted, and will return an error if we get garbage input.

in main()

```
while (fgets(s, MAXLEN, fp) != NULL) {
   char *endptr;
   long num = strtol(s, &endptr, 10); // Convert string to long with error detection

   // Check if conversion was successful
   if (*endptr != '\n' && *endptr != '\0') {
       printf("Error: Invalid input detected: %s\n", s);
       continue;  // Skip invalid entries
   }

   // Allocate memory for new list node
   x = (Link*) malloc(sizeof(Link));
   if (x == NULL) {
       printf("Error: malloc failed\n");
       exit(1);
   }

   x->data = (int)num; // Store valid integer
   x->next = head;
   head = x;
}
```

To fix infinite loops, pointer handling, and duplicate values, we'll correct the logic of bubble sort. This corrected portion of the code will now prevent unnecessary swaps if a list is sorted and terminates the swapping using lptr. This will prevent infinite loops.

Swapping logic in sort()

```
Link* sort(Link* x) {
   if (x == NULL || x->next == NULL)
       return x; // Handle empty or single-node lists

   int swapped;
   Link *ptr, *lptr = NULL;

   do {
       swapped = 0;
       ptr = x;
```

```
        while (ptr->next != lptr) {
            if (ptr->data > ptr->next->data) {  // Only swap if out of order
                int temp = ptr->data;
                ptr->data = ptr->next->data;
                ptr->next->data = temp;
                swapped = 1;
            }
            ptr = ptr->next;
        }
        lptr = ptr; // Reduce traversal range
    } while (swapped);

    return x;
}
```